Demonstrating a Realistic IP Mission Prototype

James Rash, Arturo B. Ferrer
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
Nancy Goodman, Samira Ghazi-Tehrani, NASA/GSFC
Joe Polk, Vantage Systems, Inc.
Lorin Johnson, Interface and Control Systems, Inc.
Greg Menke, Bill Miller Raytheon
Ed Criscuolo, Keith Hogie, Ron Parise
Computer Sciences Corp
7700 Hubble Dr.
Lanham-Seabrook, MD 20706

***Abstract*—** Flight software and hardware and realistic space communications environments were elements of recent demonstrations of the Internet Protocol (IP) mission concept in the lab. The Operating Missions as Nodes on the Internet (OMNI) Project and the Flight Software Branch at NASA/GSFC collaborated to build the prototype of a representative space mission that employed unmodified off-the-shelf Internet protocols and technologies for end-to-end communications between the spacecraft/instruments and the ground system/users. The realistic elements used in the prototype included an RF communications link simulator and components of the TRIANA mission flight software and ground support system. A web-enabled camera connected to the spacecraft computer via an Ethernet LAN represented an on-board instrument creating image data. In addition to the protocols at the link layer (HDLC), transport layer (UDP, TCP), and network (IP) layer, a reliable file delivery protocol (MDP) at the application layer enabled reliable data delivery both to and from the spacecraft. The standard Network Time Protocol (NTP) performed on-board clock synchronization with a ground time standard. The demonstrations of the prototype mission illustrated some of the advantages of using Internet standards and technologies for space missions, but also helped identify issues that must be addressed. These issues include applicability to embedded real-time systems on flight-qualified hardware, range of applicability of TCP, and liability for and maintenance of commercial off-the-shelf (COTS) products. The NASA Earth Science Technology Office (ESTO) funded the collaboration to build and demonstrate the prototype IP mission.

TABLE OF CONTENTS

1. INTRODUCTION

The goal of the Space Internet Technology Testbed effort was to demonstrate in the laboratory the operation of a realistic IP mission prototype as a means to investigate the viability of using standard Internet protocols, open source real time operating systems (RTOSs), and standard Ethernet as the spacecraft bus. The desire to reduce future spacecraft development and mission operations costs motivated the activity.

This activity demonstrated end-to-end satellite data flow concepts in a realistic space and ground system hardware/software environment. The demonstrated data flows included 1) automated store and forward flow of simulated science data from a spacecraft instrument to multiple end user workstations, 2) automated real time flow of housekeeping data from the spacecraft Command & Data Handling (C&DH) software to an Integrated Test and Operations System (ITOS) ground support workstation, and 3) real time data flow of ground commands from the ITOS workstation to the spacecraft instrument via the spacecraft C&DH software. This activity also investigated the Network Time Protocol (NTP) for automated time synchronization between the ITOS ground workstation and the spacecraft C&DH software, and between the spacecraft C&DH software and the simulated science instrument.

2. OVERVIEW OF TESTBED

Ground and Flight Segments

The laboratory setup consisted of a ground segment and a flight segment configured to simulate a complete mission environment.

The ground segment consisted of a Sun Ultra 10 running the ITOS ground system, a Cisco 1601/1603 router configured in multicast mode, two Principal Investigator workstations, and supporting Ethernet networking equipment.

The flight segment consisted of a C&DH component running on a MCP750 single board computer in a Motorola CPX2000 Series 6U Compact PCI Chassis, a Canon Communication Camera, model VC-C1 MK II,Camera (model VC-C1 MK II) connected to an Axis 240 Camera Server simulating a spacecraft instrument, a communication up/down link consisting of a Cisco 1601/1603 router, an Adtech Model SX/13 Data Channel Simulator, and supporting Ethernet networking equipment.

There were also two workstations (not shown in Figure 1 below), one on the Spacecraft Segment and one on the Ground Segment, both running the EtherPeek tool by WildPackets, Inc., to help with debugging and testing.

C&DH Components

The starting point for the effort was an existing partial port of the Triana C&DH flight software rehosted from VxWorks to the Linux operating system. This port included TCP/IP interfaces between the Triana flight software on a Power PC 603 target with Ethernet interfaces to the ITOS workstation.

Since generic Linux uses a time sliced, round robin scheduling mechanism, which is not capable of servicing real time deadlines typically found in flight software implementations, the Triana software was rehosted to a Real Time Operating System (RTOS).

The initial investigation identified Real-Time Embedded Multiprocessing System (RTEMS) for use on the testbed (http://www.rtems.com/). RTEMS is open source and portable, and Code 582 has significant experience with the product. Recognizing a 6-month time constraint and limited personnel resources, the team initiated a procurement with Online Applications Research, Inc., Huntsville, AL, to add the Multicast Dissemination Protocol (MDP) client and server, and the Network Time Protocol (NTP) client and server, to the RTEMS core capability. Due to procurement delays, it was necessary to identify a backup RTOS, resulting in the use of RTLinux from FSMLabs.

RTLinux is an open source product that runs on top of Linux, so it not only has the capabilities of Linux, but also can service real time interrupts in a timely manner. A detailed treatment of the structure and capabilities of RTLinux is beyond the scope of this discussion. The reader is invited to seek more information from FSMLabs, Inc. (http://www.fswlabs.com/)

The testbed team retargeted the Triana flight software to the RTLinux RTOS. The heritage Triana software interprocess communication mechanism, the Software Bus, used information in CCSDS packet headers to route command and telemetry packets between software subsystems onboard. Considering time and resource constraints, the team decided to keep the CCSDS packet structure, allowing reuse of the Software Bus and the existing Triana ITOS command and telemetry database with minor modifications to both.

Science data transfer from the instrument to the C&DH, and between the C&DH and ground was implemented in UDP packets.

The flight software components are discussed below:

- Command Ingest (CI) - Flight software component used to accept, validate and distribute real-time CCSDS command packets received from the ITOS ground workstation. CI was modified from the flight VxWorks version to use an IP socket and port. It received TCP/IP packets from ITOS and extracted the CCSDS packets in the data field.
- Telemetry Output (TO) - Flight software component used to wrap housekeeping CCSDS packets in IP headers for transmission to the ITOS ground workstation. TO was modified from the flight VxWorks version to use an IP socket and port. It received CCSDS packets from software subsystems onboard and encapsulated the CCSDS packets in IP headers for transmission to the ITOS GSE workstation.
- Health and Safety (HS) - Flight software component used to request, collect and format health and safety data from the flight software subsystem tasks. It sends housekeeping requests to the subsystem tasks, collects the responses into one CCSDS packet, then sends the CCSDS packet to TO for downlink.
- Software Bus (SB) – Flight software component used for interprocess communications. SB routes CCSDS packets based on application ID numbers and validates sender and receiver tasks. SB is based on a table driven routing mechanism using message queues provided by the RTOS.
- Instrument Support (IS) – This is a new component added for this effort. IS received instrument commands from the CI task delivered through SB. The instrument command was packaged in a CCSDS packet to accommodate the SB routing mechanism. IS then extracted the instrument command from the data portion of the CCSDS packet and forwarded the command via a socket/Ethernet interface to the simulated science instrument. IS received UDP science data packets from the instrument and placed the data in a specified directory in file format.
- File Transfer (FT) – This is a new component added for this effort. FT polled a directory for the creation of new files. When it finds a new file, it sends the file to the ground using the MDP protocol.

Deleted: <sp>

Ground Components

ITOS is the ground support environment used for spacecraft commanding and telemetry display. It runs on a Sun Ultra 10. The ITOS database from the Triana mission was used as a starting point and modified to accommodate housekeeping telemetry from the IS task and to include instrument commands. Instrument commands included commands to turn the camera on and off, adjust the picture size and resolution, and adjust the picture frequency. A display page for real-time telemetry from the IS task was added.

The Principal Investigator Workstations were two desktop computers running RedHat Linux and the xview to display JPEG picture files.

3. SPACE RELATED ISSUES

Flight Software versus Desktop Applications

The spacecraft flight software domain is very different from industrial and desktop applications. While there is much to be learned and can be applied to spacecraft flight software from these areas, it has to be realized that, at least for GSFC flight software, many seemingly advantages of COTS software products may not actually be applicable. This can mostly be attributed to the fact that GSFC spacecraft hardware usually deviate in some manner from the COTS hardware platforms targeted by COTS software products, requiring some customization of the COTS software product that voids technical support. For example, the GSFC board support package written to support the port of VxWorks to the Mongoose V processor, resulted in a situation that reduced the value of technical support from the vendor.

Further, consider the responsibilities of the flight software branch, Code 582. Code 582 is tasked with developing, maintaining and operating software for embedded space applications. This includes troubleshooting in-flight software and hardware anomalies. To perform these duties, a detailed knowledge is required of the hardware/software interfaces, and of most software components. COTS software products are attractive in reducing development costs, but may increase maintenance costs, because the purchase of the source code and additional personnel resources may be required to understand the internal workings of the product.

Current GSFC flight software implementations use a real-time embedded software model implemented by VxWorks and RTEMS, where the Real Time Operating System (RTOS) provides interrupt vectoring for software responses to hardware events, and pre-emptive multitasking. The interrupt services have been critical in past flight applications for software to meet absolute deadlines where failure to respond within a given time limit can cause system failure. On the GLAS instrument for the ICESat mission, this hard real time limitation was 25 milliseconds. When an interrupt occurs, software tasking is paused, the processor context is saved, execution is switched to a specific interrupt service routine, the interrupt is serviced, then the processor context is switched back to the previous state of software tasking. These interrupts can set semaphores which can trigger the execution of high priority tasks. Pre-emptive multitasking permits the execution of multiple software programs to run as separate tasks within a single larger software environment. A lower priority task is allowed to execute until it completes its function, or is either preempted by an interrupt or a higher priority task that has encountered a condition that causes it to run. It is this interrupt and preemption scenario that is unique to real time embedded systems and is NOT addressed by desktop applications or most desktop operating systems.

Desktop operating systems such as Linux in particular were not designed to service interrupts and allow for task preemption. Linux follows a process model, where software components (processes) are given equal periods of time to execute. No preemption is allowed, and interrupt response time can be extremely variable. Such operating systems are generally not suitable for representative real time applications requiring interrupt driven data transfers such as 1553 bus applications, attitude control systems or the servicing of sensor driven events.

To accommodate this limitation of Linux, the RTLinux product of FSMLabs was chosen for use in this effort. RTLinux provides the ability to service hard real time deadlines, and permits Linux OS functionality. Unfortunately, the real time performance of RTLinux was not measured or demonstrated. Nonetheless, the embedded prioritized task model was implemented within a single RTLinux process. Essentially this effort executed flight software in a "window" type process. The effect is that the flight software is now just a timesliced process running on the Linux operating system. RTLinux permits interrupt servicing and runs the entire Linux OS as a preemptable process. If one considers this change in model, there is a definite transition where the "flight software" no longer controls a vast majority of system resources. These implications for hard real time behavior need to be characterized.

4. Lessons Learned

RTOS Ports

The intended hardware target was the Motorola MCP750 because existing ports were available for RTEMS, and it was estimated that RTLinux could be ported to this architecture in a period of a week. However, the RTLinux rehost to the MCP750 was much more involved than expected and eventually required assistance from FSMLabs for completion. The apparent difficulty was rehosting a Linux kernel. In this case, the MCP750 was not a custom board, but it was not a widely supported platform, and the details were beyond our knowledge base.

RTEMS is functional on the MCP750. However, due to time and resource constraints OAR was hired to implement MDP and NTP in the RTEMS core. The implementation was delivered, but is currently unverified.

An additional difference between RTEMS and RTLinux is the memory resources required for each product. RTEMS requires 400K bytes compressed on disk while RTLinux requires 1.4 Mbytes. The RAM footprint is significantly different as well. RTEMS requires 470Kbytes while RTLinux is dynamically sized tending to expand to a minimum of 8 Mbytes. These characteristics indicate that RTEMS is targeted for embedded systems and requires minimal resources. RTLinux is a convert from the desktop world and requires more resources for operation.

From this effort, it was concluded that operating system expertise is necessary for any endeavor, particularly for IP software such as MDP and NTP that require standard Linux OS calls and libraries. Operating system expertise is an essential core technical skill that is frequently taken for granted or overlooked.

Multicast Dissemination Protocol (MDP)

A primary design goal of MDP is to provide a reliable multicast protocol approach that is suitable for reliable dissemination of data over both wireless and wired networks. For this effort, the team used MDP version 2 downloaded from the internet. The MDP project homepage is http://mdp.pf.itd.nrl.navy.mil/.

The flight software process had an MDP server implemented as a thread. This demonstrated viability for implementation within the existing embedded system flight software model. The MDP thread implementation was very successful. It continually scans a "hot" directory, and if a new file appears in that directory, the MDP server automatically sends it to the MDP client(s). The transfer is based on UDP packets and no acknowledgements (ACKs) are required. If an MDP block arrives at the client in a corrupt state, or does not arrive at all, the client sends a negative acknowledgement (NACK) to the server requesting retransmission.

ACKs are returned from the client to the server only when a file has been successfully received on the ground. This ACK can be used by the server to delete files that have been successfully transmitted. Although this option exists, it was not implemented in this demo.

The configuration of MDP in this effort was limited to a static version that cannot be interactively commanded. The setup parameters for MDP were hard coded and initiated on thread startup. Typically MDP options are entered on a command line in a process model operating system. Further work is necessary to determine if interactive commands to an MDP thread can be accommodated, or if a manner of stopping the thread, altering startup parameters, and restarting the thread is a viable option in an embedded systems environment. Additionally, coordination between the MDP client and server needs to be demonstrated during periods of uplink dropouts. It can be anticipated that the MDP server will continue to send files in the absence of ACKs and NACKs from the MDP client. If MDP client ACKs and NACKs are buffered and sent during the next spacecraft contact, how will the MDP server respond? This behavior needs to be investigated for representative periods where more time is spent out of contact than in contact with the spacecraft. Nonetheless, MDP has significant potential for space flight applications, and merits further examination. Also consider the maintenance requirement. If in-flight maintenance is required, then a closer examination is needed to understand the internals.

Network Time Protocol (NTP)

The testbed team downloaded the NTP product from the website at http://www.ntp.org, and implemented the NTP server as a separate process from the flight software. Due to time constraints the implementation of NTP as a thread was not investigated. NTP required a rather large footprint of 199K and offered very little control over when time queries were performed. Hence, for most Earth orbit missions, the spacecraft will be out of contact with the ground for significant periods and NTP time queries will be unanswered.

NTP comes as a packaged configuration for desktops and was successfully implemented on the MCP750 for this effort. Since there is little visibility in the setup process, no determination can be made on configuring NTP for custom spacecraft architectures. If COTS processor hardware is used for spacecraft, configuring NTP may not be an issue, given that a Linux or RTLinux type operating system is used. Unfortunately, the RTEMS upgrade to include NTP was not demonstrated.

Further, the NTP time synchronization algorithm assumes linear oscillator degradation. Since it is a complex algorithm, an investigation is needed to determine how it will accommodate cyclic or intermittent oscillator anomalies due to thermal and radiation effects or hardware failure. An example of a flight software fix for timing hardware failure is the TRMM “no-clock” fix effort completed in 1999 and is currently ready for insertion into the spacecraft. Such a fix for an NTP spacecraft is currently unlikely due to unfamiliarity with the NTP internals and its interaction with low level hardware. The maintenance requirement question surfaces again.

On the other hand, NTP appears to be a fair candidate for synchronizing time between spacecraft flying in constellations, or to synchronize NTP to a GPS receiver onboard a spacecraft. We can not make a definitive determination concerning NTP, since our involvement was limited to the implementation of NTP as a process, not as an embedded system thread, and since we had limited time to investigate its configuration, behavior, performance and internals.

TCP

The testbed exercise used TCP sockets for the Spacecraft C&DH and ITOS command link. On most occasions, the TCP connection was sufficient. However, there were instances where the C&DH software would lock up when the TCP connection was broken. This type of condition commonly results from improper application handling of socket error signals. Resource and time constraints prevented the development of a full socket error handling mechanism within the application, which would naturally be included in an actual flight implementation. During testbed testing activities, we recovered by rebooting both the C&DH computer and the ITOS workstation and re-establishing the TCP connection.

5. CONCLUSIONS

Demonstrations conducted using the testbed served to boost familiarity with and acceptability of IP technology for possible use on future NASA missions. The use of flight software from a NASA mission (Triana) and other realistic mission elements in the testbed ensured serious consideration of the results. Further, the involvement of the Flight Software Branch at GSFC ensured that the experience gained with IP-centric implementations could be immediately related to actual mission projects. The experience also represented a stepping stone towards adopting new approaches to mission development.

There can be no doubt concerning the benefits of standard protocols and buses. IP protocols have been demonstrated to be feasible for non-hard real time flight applications. The use of IP protocols in more rigorous time constrained environments requires further investigation, because this was not addressed in the testbed task.

Off-the-shelf protocols and toolkits such as MDP and NTP have potential for mission use but must be viewed in the larger scope of the spacecraft lifecycle. Savings in development need to be assessed relative to the effort required to gain the expertise with product internals necessary for effective maintenance and troubleshooting during both spacecraft integration and testing (I&T) and on orbit operations.

The decision to fly IP is independent of the decision to fly an on-board LAN such as Ethernet. Similarly, the decision to fly an open source real time operating system such as RTLinux is independent of the other two decisions. All of these decision need to flow from a consensus among program management, and the system, hardware, and software engineers based on specific requirements. Through such cooperation the correct combination of COTS vs. custom hardware and software can be identified and assembled for a successful mission.

6. FUTURE WORK

Despite successful laboratory and flight experiments, IP technology has not yet been carried forward into routine mission development at NASA. The near term obstacles lie

in current absence of RTOSs that support standard POSIX operating system interfaces on custom flight hardware, and the performance characterization of IP in hard real time environments.

Fortunately, the development of high-speed radiation-hardened processors for flight is rapidly reducing hardware limitations. However, niches will remain for micro-controllers and heritage hardware. RTOS selection for flight projects often goes hand-in-hand with flight hardware selection. Modern operating systems need modern hardware. The trend to provide these resources for flight use is slow, but gaining momentum. However, one may expect that for some embedded spacecraft applications, the migration to IP technology will be difficult.

The CHIPSat mission (http://chips.ssl.berkeley.edu/) and the CANDOS experiment that flew on Shuttle mission STS-107 (http://ipinspace.gsfc.nasa.gov/CANDOS/) took IP technology to the flight demonstration level in the NASA context, although the first IP mission that was designed as such (AlSAT-1) was flown by Surrey Satellite Technology (SSTL) in November, 2002 (http://www.sstl.co.uk/).

These mission experiences, and the collaboration between the GSFC Flight Software Branch and the GSFC Operating Missions as Nodes on the Internet (OMNI) Project (http://ipinspace.gsfc.nasa.gov/), influenced the planned implementation of IP technologies in the Global Precipitation Measurement (GPM) mission, which will fly in 2008 (http://gpm.gsfc.nasa.gov/).

The GPM project has baselined both IP space-to-ground communications and an onboard Ethernet network using the flight Ethernet cards and switch technology developed by GSFC Code 561. A major issue to be resolved is reliable and redundant LAN configurations to insure fault tolerance and failover capabilities.

Familiarity with space mission requirements and experience with IP mission prototypes support the expectation that the UDP transport layer protocol would be required for basic data communications on the space-ground link. While TCP can be used under some conditions to meet some mission requirements, it is not generally sufficient for spacecraft commanding,. This is immediately clear. First, commanding must be possible even when, as is often the case, there is only a forward link to, but no return link from, the spacecraft. Since TCP requires two-way communications, TCP cannot suffice. Additional factors that affect the viability of TCP for space communications are bit error rate (BER), link latency (bandwidth-delay product), and link asymmetry. Since UDP does not require two-way communications, and is unaffected by link asymmetry or link latency, it becomes the natural alternative to TCP for most IP mission data communications requirements. In particular, it is clear that it will suffice for the GPM baselined requirements. Whether TCP could be advantageously included for GPM in addition to the planned UDP mechanisms is yet to be determined.

IP mission requirements for reliable file delivery can be met by an application running over UDP. Two candidate applications are available: CCSDS File Delivery Protocol (CFDP) and Multicast Dissemination Protocol (MDP). CFDP and MDP are similar but differ in important respects. The testbed demonstrated the use of MDP, but not CFDP. Evaluations of the two applications are being conducted, although a preliminary assessment by Code 582 has given the edge to CFDP for use on GPM.

While much work has been done, the engineering of hard real time systems, where missed deadlines for software responses may cause system failure, to use IP technology has not yet been carried forward into routine mission development at NASA. Certainly implementing IP technology in nano-satellites and high performance instruments would be challenging due to restricted system resources and rigorous execution deadlines. Future work lies in the characterization of IP technology using the embedded system RTOS model in varying hard real time performance ranges.

7. ACKNOWLEDGEMENTS

NASA's Earth Science Technology Office (ESTO) funded the research described in this paper.

James Rash *– Goddard Space Flight Center – Mr. Rash currently manages the Operating Missions as Nodes on the Internet (OMNI) project in the Advanced Architectures and Automation Branch at NASA's Goddard Space Flight Center. He also leads development of formal methods capabilities with respect to agent-based systems. Previous assignments have included development of systems applying artificial intelligence and evolutionary programming techniques. He was Principal Investigator on the IP mission testbed task.*

Arturo Ferrer *– Goddard Space Flight Center – Mr. Ferrer is the lead engineer for the Flight Software Technology Lab in the Flight Software Branch at NASA's Goddard Space Flight Center. He also has development responsibilities for a number of software subsystems on in-house projects and provides technical support for out-of-house flight projects. He was the Co-Investigator on the IP mission testbed task.*